

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Applicant:

Christopher H. Genly

Serial No.: 09/494,796

Filed: January 31, 2000

For: Providing Information in  
Response to Spoken Requests

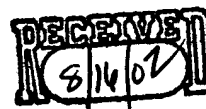
§  
§  
§  
§  
§  
§  
§  
§

Art Unit: 2654

Examiner: Angela Armstrong

Atty Docket: ITL.0343-US  
P8394

Board of Patent Appeals & Interferences  
Commissioner for Patents  
Washington, D.C. 20231



**APPEAL BRIEF**

Sir:

Applicant respectfully appeals from the final rejection mailed June 18, 2002.

**I. REAL PARTY IN INTEREST**

The real party in interest is the assignee Intel Corporation.

**II. RELATED APPEALS AND INTERFERENCES**

None.

**III. STATUS OF THE CLAIMS**

Claims 1-30 are rejected. Each rejection is appealed.

08/14/2002 VTOWLER 00000001 09494796

01 FC:120

320.00 OP

Date of Deposit: August 7, 2002  
I hereby certify under 37 CFR 1.8(a) that this correspondence is being deposited with the United States Postal Service as **first class mail** with sufficient postage on the date indicated above and is addressed to the Commissioner for Patents, Washington DC 20231.  
Cynthia L. Hayden  
Cynthia L. Hayden

RECEIVED  
BOARD OF PATENT APPEALS  
AND INTERFERENCES  
AUG 12 PM 2:25

#### **IV. STATUS OF AMENDMENTS**

There were no amendments.

#### **V. SUMMARY OF THE INVENTION**

An application may respond to conversational speech, with spoken or visual responses including using graphical user interfaces, in accordance with one embodiment of the present invention. In some embodiments of the present invention, a limited domain may be utilized to increase the accuracy of speech recognition. A limited or small domain allows focused applications to be implemented wherein the recognition of speech is improved because the vocabulary is limited.

A variety of techniques may be utilized for speech recognition. However, in some embodiments of the present invention, the process may be simplified by using surface parsing. In surface parsing questions or statements are handled separately and there is no movement to convert questions into the same subject, verb, object order as a statement. As a result, conventional, commercially available software may be utilized for some aspects of speech recognition with surface parsing. However, in some embodiments of the present invention, deep parsing with movement may be more desirable. See specification at page 3, lines 6 through 24.

As used herein, the term "conversational" as applied to a speech responsive system involves the ability of the system to respond to broadly or variously phrased requests, to use conversational history to develop the meaning of pronouns, to track topics as topics change and to use reciprocity. Reciprocity is the use of some terms that were used in the questions as part of the answer.

In any case, the system uses a voice user interface (VUI) which interfaces between the spoken request for information from the user and the system. The voice user interface and a

graphical user interface advantageously communicate with one another so that each knows any inputs that the other has received. That is, if information is received from the graphical user interface to provide focus to a particular topic, such as a television program, this information may be provided to the voice user interface to synchronize with the graphical user interface. This may improve the ability of the voice user interface to respond to requests for information since the system then is fully cognizant of the context in which the user is speaking.

The voice user interface may include a number of different states including the show selected, the audio volume, pause and resume and listen mode. The listen mode may include three listening modes: never, once and always. The never mode means that the system is not listening and the speech recognizer is not running. The once mode means that the system only listens for one query. After successfully recognizing a request, it returns to the never mode. The always mode means that the system will always listen for queries. After answering one query, the system starts listening again. See specification at page 4, line 1 through page 5, line 4.

A listen state machine utilized in one embodiment of the present invention may reflect whether the system is listening to the user, working on what the user has said or has rejected what the user has said. A graphical user interface may add itself as a listener to the listen state machine so that it may reflect the state to the user. There are four states in the listen state machine. In the idle state, the system is not listening. In the listening state, the system is listening to the user. In the working state, the system has accepted what the user has said and is starting to act on it. In the rejected state, what the user said has been rejected by the speech recognition engine.

Referring to Figure 1, the system software may include an application 16 that may be an electronic programming guide application in one embodiment of the present invention. In the

illustrated embodiment, the application 16 includes a voice user interface 12 and a graphical user interface 14. The application 16 may also include a database 18 which provides information such as the times, programs, genre, and subject matter of various programs stored in the database 18. The database 18 may receive inquiries from the voice user interface 12 and the graphical user interface 14. The graphical and voice user interfaces may be synchronized by synchronization events. See specification at page 5, line 5 through page 6, line 2.

The voice user interface 12 may also include a speech synthesizer 20, a speech recognizer 21 and a natural language understanding (NLU) unit 10. In other embodiments of the present invention, output responses from the system may be provided on a display as text rather than as voice output responses from a synthesizer. The voice user interface 12 may include a grammar 10a which may be utilized by the recognizer 21.

A state vector is a representation of the meaning of an utterance by a user. A state vector may be composed of a set of state variables. Each state variable has a name, a value and two flags. An in-context state vector may be developed by merging an utterance vector which relates to what the user said and a history vector. A history vector contains information about what the user said in the past together with information added by the system in the process of servicing a query. Thus, the in-context state vector may account for ambiguity arising, for example, from the use of pronouns. The ambiguity in the utterance vector may be resolved by resorting to a review of the history vector and particularly the information about what the user said in the past.

In any state vector, including utterance, history or in-context state vectors, the state variables may be classified as SELECT or WHERE variables (borrowing the terms SELECT and WHERE from the SQL database language). SELECT variables represent information a user is

requesting. In other words, the SELECT variable defines what the user wants the system to tell the user. This could be a show time, length or show description, as examples.

WHERE variables represent information that the user has supplied. A WHERE variable may define what the user has said. The WHERE variable provides restrictions on the scope of what the user has asked for. Examples of WHERE variables include show time, channel, title, rating and genre. See specification at page 6, line 3 through page 7, line 4.

The query "When is X-Files on this afternoon?" may be broken down as follows:

Request: When (from "When is X-Files on this afternoon?")

Title: X-Files

Part\_of\_day\_range: afternoon

The request (when) is the SELECT variable. The WHERE variables include the other attributes including the title (X-Files) and the time of day (afternoon).

The information to formulate responses to user queries may be stored in a relational database in one embodiment of the present invention. A variety of software languages may be used. By breaking a query down into SELECT variables and WHERE variables, the system is amenable to programming in well known database software such as Structured Query Language (SQL). SQL is standard language for relational database management systems. In SQL, the SELECT variable selects information from a table. Thus, the SELECT command provides the list of column names from a table in a relational database. The use of a WHERE command further limits the selected information to particular rows of the table. Thus, a bare SELECT command may provide all the rows in a table and the combination of a SELECT and a WHERE command may provide less than all the rows of a table, including only those items that are responsive to both the SELECT and the WHERE command. Thus, by resolving spoken queries

into SELECT and WHERE variables, the programming may be facilitated in some embodiments of the present invention.

Referring to Figure 2 a user request or query 26 may result in a state vector 30 with a user flag 34 and a grounding flag 32. The user flag 34 indicates whether the state variable originated from the user's utterance. The grounding flag 32 indicates if the state variable has been grounded. A state variable is grounded when it has been spoken by the synthesizer to the user to assure mutual understanding. The VUI 12 may repeat portions of the user's query back to the user in its answer. See specification at page 7, line 5 through page 8, line 9.

Grounding is important because it gives feedback to the user about whether the system's speech recognition was correct. For example, consider the following spoken interchange:

1. User: "Tell me about X-Files on Channel 58".
2. System: "The X-Files is not on Channel 50".
3. User: "Channel 58".
4. System: "On Channel 58, an alien..."

At utterance number 1, all state variables are flagged as from the user and not yet grounded. Notice that the speech recognizer confused fifty and fifty-eight. At utterance number 2, the system has attempted to repeat the title and the channel spoken by user and they are marked as grounded. The act of speaking parts of the request back to user lets the user know whether the speech recognizer has made a mistake. Grounding enables correction of recognition errors without requiring re-speaking the entire utterance. At utterance number 3, the user repeats "58" and the channel is again ungrounded. At utterance number 4, the system speaks the correct channel and therefore grounds it.

Turning next to Figure 3, software 36 for speech recognition involves the use of an application program interface (API) in one embodiment of the present invention. For example, the JAVA speech API may be utilized in one embodiment of the present invention. Thus, as indicated in block 38, initially the API recognizes an utterance as spoken by the user. The API then produces tags as indicated in block 40. These tags are then processed to produce the state vector as indicated in block 42. See specification at page 8, line 10 through page 9, line 9.

Upon recognizing an utterance, the JAVA speech API recognizer produces an array of tags. Each tag is a string. These strings do not represent the words the user spoke but instead they are the strings attached to each production rule in the grammar. These tags are language independent strings representing the meaning of each production rule. For example, in a time grammar, the tags representing the low order minute digit may include text which has no meaning to the recognizer. For example, if the user speaks "five", then the recognizer may include the tag "minute: 5" in the tag array.

The natural language understanding (NLU) unit 10 develops what is called an in-context meaning vector 48 indicated in Figure 4. This is a combination of the utterance vector 44 developed by the recognizer 21 together with the history vector 46. The history vector includes information about what the user said in the past together with information added by the system in the process of servicing a query. The utterance vector 44 may be a class file in an embodiment using JAVA. The history vector 46 and a utterance vector 44 may be merged by structural history management software 62 to create the in-context meaning vector 48. The history, utterance and in-context meaning vectors are state vectors.

The in-context meaning vector 48 is created by decoding and replacing pronouns which are commonly used in conversational speech. The in-context meaning vector is then used as the

new history vector. Thus, the system decodes the pronouns by using the speech history vector to gain an understanding of what the pronouns mean in context.

The in-context meaning vector 48 is then provided to dialog control software 52. The dialog control software 52 uses a dialog control file to control the flow of the conversation and to take certain actions in response to the in-context meaning vector 48. See specification at page 9, line 10 through page 10, line 13.

These actions may be initiated by an object 51 that communicates with the database 18 and a language generation module 50. Prior to the language generation module 50, the code is human language independent. The module 50 converts the code from a computer format to a string tied to a particular human understood language, like English. The actions object 51 may call the synthesizer 20 to generate speech. The actions object 51 may have a number of methods (See Table I infra).

Thus, referring to Figure 5, the dialog control software 52 initially executes a state control file by getting a first state pattern as indicated in block 54 in one embodiment of the present invention. Dialog control gives the system the ability to track topic changes.

The dialog control software 52 uses a state pattern table (see Table I below). Each row in the state pattern table is a state pattern and a function. The in-context meaning vector 48 is compared to the state pattern table one row at a time going from top to bottom (block 56). If the pattern in the table row matches the state vector (diamond 58), then the function of that row is called (block 60). The function is also called a semantic action.

Each semantic action can return one of three values: CONTINUE, STOP and RESTART as indicated at diamond 61. If the CONTINUE value is returned, the next state pattern is obtained, as indicated at block 57, and the flow iterates. If the RESTART value is returned, the



system returns to the first state pattern (block 54). If the STOP value is returned, the system's dialog is over and the flow ends.

The action may do things such as speak to the user and perform database queries. Once a database query is performed, an attribute may be added to the state vector which has the records returned from the query as a value. Thus, the patterns consist of attribute, value pairs where the attributes in the state pattern table correspond to the attributes in the state vector. The values in the pattern are conditions applied to the corresponding values in the state vector. See specification at page 10, line 14 through page 11, line 18.

Table I

1	Request	Title	Channel	Time	nfound	function
2	Help					giveHelp
3	Tv_on					turnOnTV
4	Tv_off					turnOffTV
5	tune		exists			tuneTV
6				not exists		defaultTime
7						checkDBLimits
8						queryDB
9					0	relaxConstraints
10					-1	queryDB
11					0	saySorry
12					1	giveAnswer
13					>1	giveChoice

Thus, in the table above, the state patterns at lines 2-5 are basic functions such as help, turn the television on or off and tune the television and all return a STOP value.

In row six, the state pattern checks to see if the time attribute is defined. If not, it calls a function called defaultTime() to examine the request, determine what the appropriate time should be, set the time attribute, and return a CONTINUE value.

In row seven, the pattern is empty so the function `checkDBLimits()` is called. A time range in the user's request is checked against the time range spanned by the database. If the user's request extends beyond the end of the database, the user is notified, and the time is trimmed to fit within the database range. A `CONTINUE` value is returned.

Row eight calls the function `queryDB()`. `QueryDB()` transforms the state vector into an SQL query, makes the query, and then sets the `NFOUND` variable to the number of records retrieved from the database. The records returned from the query are also inserted into the state vector. See specification at page 12, line 1 through page 13, line 4.

At row nine a check determines if the query done in row eight found anything. For example, the user may ask, "When is the X-Files on Saturday?", when in fact the X-Files is really on Sunday. Rather than telling the user that the X-Files is not on, it is preferable that the system say that "the X-Files is not on Saturday, but is on Sunday at 5:00 p.m". To do this, the constraints of the user's inquiry must be relaxed by calling the function `relaxConstraints()`. This action drops the time attribute from the state vector. If there were a constraint to relax, `relaxConstraints()` sets `NFOUND` to -1. Otherwise, it leaves it at zero and returns a `CONTINUE` value.

Row 10 causes a query to be repeated once the constraints are relaxed and returns a `CONTINUE` value. If there were no records returned from the query, the system gives up, tells the user of its failure in row 11, and returns a `STOP` value. In row 12 an answer is composed for the user if one record or show was found and a `STOP` value is returned.

In row 13, a check determines whether more than one response record exists. Suppose X-Files is on both channels 12 and 25. `GiveChoice()` tells the user of the multiple channels and asks the user which channel the user is interested in. `GiveChoice()` returns a `STOP` value

(diamond 61, Figure 5), indicating that the system's dialog turn is over. If the user tells the system a channel number, then the channel number is merged into the previous inquiry stored in history.

The system tracks topic changes. If the user says something that clears the history, the state pattern table simply responds to the query according to what the user said. The state pattern table responds to the state stored in the in-context vector. See specification at page 13, line 5 through page 14, line 4.

Turning next to Figure 6, the software 62 implements structural history management (SHM). Initially the flow determines at diamond 64 whether an immediate command is involved. Immediate commands are utterances that do not query the database but instead demand immediate action. They do not involve pronouns and therefore do not require the use of structural history. An example would be "Turn on the TV". In some cases, an immediate command may be placed between other types of commands. The immediate command does not effect the history. This permits the following sequence of user commands to work properly:

1. "When is X-Files on",
2. "Turn on the TV",
3. "Record it".

The first sentence puts the X-Files show into the history. The second sentence turns on the television. Since it is an immediate command, the second sentence does not erase the history. Thus, the pronoun "it" in the record command (third sentence) can be resolved properly.

Thus, referring back to Figure 6, if an immediate command is involved, the history is not changed as indicated in block 66. Next, a check at diamond 68 determines whether a list selection is involved. In some cases, a query may be responded to with a list of potential shows and a request that the user verbally select one of the listed shows. The system asks the user

which title the user is interested in. The user may respond that it is the Nth title. If the user utterance selects a number from a list, then the system merges with history as indicated in block 70. Merging with history refers to an operation in which the meaning derived from the speech recognizer is combined with history in order to decode implicit references such as the use of pronouns.

Next, a check at diamond 72 determines whether the query includes both SELECT and WHERE variables. If so, history is not needed to derive the in-context meaning as indicated in block 74. See specification at page 14, line 5 through page 15, line 8.

Otherwise, a check determines whether the utterance includes only SELECT (diamond 76) or only WHERE (diamond 80) variables. If only a SELECT variable is involved, the utterance vector is merged with the history vector (block 78).

Similarly, if the utterance includes only a WHERE variable, the utterance is merged with history as indicated in block 82. If none of the criteria set forth in diamonds 64, 68, 72, 76 or 80 apply, then the history is not changed as indicated in block 84.

As an example, assume that the history vector is as follows:

Request: When (from "When is X-Files on this afternoon?")  
Title: X-Files  
Part\_of\_day\_range: afternoon.

Thus the history vector records a previous query "When is X-Files on this afternoon?".

Thereafter, the user may ask "What channel is it on?" which has the following attributes:

Request: Channel (from "What channel is it on?")

Thus, there is a SELECT attribute but no WHERE attribute in the user's query. As a result, the history vector is needed to create an in-context or merged meaning as follows:

Request: Channel (from "What channel is X-Files on this afternoon?")  
Title: X-Files  
Part\_of\_day\_range: afternoon.

Notice that the channel request overwrote the when request. See specification at page 15, line 9 through page 16, line 7.

As another example, assume the history vector includes the question "What is X-Files about?" which has the following attributes:

Request: About (from "What is X-Files about?")  
Title: X-Files

Assume the user then asks "How about Xena?" which has the following attributes:

Title: Xena (from "How about Xena?")

The query results in an in-context meaning as follows when merged with the history vector:

Request: About (from "What is Xena about?")  
Title: Xena.

Since there was no SELECT variable obtainable from the user's question, the SELECT variable was obtained from the historical context (i.e. from the history vector). Thus, in the first example, the WHERE variable was missing and in the second variable the SELECT variable was missing. In each case the missing variable was obtained from history to form an understandable in-context meaning.

If an utterance has only a WHERE variable, then the in-context meaning vector is the same as the history vector with the utterance's WHERE variable inserted into the history vector. If the utterance has only a SELECT variable, then the in-context meaning is the same as the history vector with the utterance's SELECT variable inserted into the history vector. If the utterance has neither a SELECT or a WHERE variable, then the in-context meaning vector is the

same as the history vector. If the utterance has both parts, then the in-context meaning is the same as that of the utterance and the in-context meaning vector becomes the history vector.

The software 86, shown in Figure 7, coordinates actions between the graphical user interface and the voice user interface in one embodiment of the invention. A show is a television show represented by a database record. A show is basically a database record with attributes for title, start time, end time, channel, description, rating and genre.

More than one show is often under discussion. A collection of shows is represented by a ShowSet. The SHOW\_SET attribute is stored in the meaning vector under the SHOW\_SET attribute. If only one show is under discussion, then that show is the SHOW\_SET. See specification at page 16, line 8 through page 17, line 18.

If the user is discussing a particular show in the SHOW\_SET, that show is indicated as the SELECTED\_SHOW attribute. If the attribute is -1, or missing from the meaning vector, then no show in the SHOW\_SET has been selected. When the voice user interface produces a ShowSet to answer a user's question, SHOW\_SET and SELECTED\_SHOW are set appropriately. When a set of shows is selected by the graphical user interface 14, it fires an event containing an array of shows. Optionally, only one of these shows may be selected. Thus, referring to diamond 88, if the user selects a set of shows, an event is fired as indicated in block 90. In block 92, one of those shows may be selected. When the voice user interface 12 receives the fired event (block 94), it simply replaces the values of SHOW\_SET and SELECTED\_SHOW (block 96) in the history vector with those of a synchronization event.

When the voice user interface 12 translates a meaning vector into the appropriate software language, the statement is cached in the history vector under the attributes. This allows unnecessary database requests to be avoided. The next time the history vector is translated, it is

compared against the cached value in the history vector. If they match, there is no need to do the time consuming database query again.

The conversational model 100 (Figure 8) implemented by the system accounts for two important variables in obtaining information about television programming: time and shows. A point in time may be represented by the a JAVA class calendar. A time range may be represented by a time range variable. The time range variable may include a start and end calendar. The calendar is used to represent time because it provides methods to do arithmetic such as adding hours, days, etc. See specification at page 17, line 19 through page 18, line 18.

The time range may include a start time and end time either of which may be null indicating an open time range. In a state vector, time may be represented using attributes such as a WEEK\_RANGE which includes last, this and next; DAY\_RANGE which includes now, today, tomorrow, Sunday, Monday. . ., Saturday, next Sunday. . ., last Sunday. . ., this Sunday. . .; PART\_OF\_DAY\_RANGE which includes this morning, tonight, afternoon and evening; HOUR which may include the numbers one to twelve; MINUTE which may include the numbers zero to fifty-nine; and AM\_PM which includes AM and PM.

Thus, the time attributes may be composed to reflect a time phase in the user's utterance. For example, in the question, "Is Star Trek on next Monday at three in the afternoon?" may be resolved as follows:

Request: When  
Title: Star Trek  
Day\_Range: Next Monday  
Part\_of\_Day\_Range: Afternoon  
Hour: 3

Since the state vector is a flat data structure in one embodiment of the invention, it is much simpler and uses simpler programming. The flat data structure is made up of attribute,

value pairs. For example, in the query "When is X-Files on this afternoon?" the request is the "when" part of the query. The request is an attribute whose value is "when". Similarly, the query has a title attribute whose value is the "X-Files". Thus, each attribute, value pair includes a name and a value. The data structure may be simplified by ensuring that the values are simple structures such as integers, strings, lists or other database records as opposed to another state vector. See specification at page 18, line 19 through page 19, line 19.

In this way, the state vector contains that information needed to compute an answer for the user. The linguistic structure of the query, such as whether it is a phrase, a clause or a quantified set, is deliberately omitted in one embodiment of the invention. This information is not necessary to compute a response. Thus, the flat data structure provides that information and only that information needed to formulate a response. The result is a simpler and more useful programming structure.

The software 116 for creating the state vector, shown in Figure 8A in accordance with one embodiment of the present invention, receives the utterance as indicated in block 117. An attribute of the utterance is determined as indicated in block 118. A non-state vector value is then attached to the attribute, value pair, as indicated in block 119.

Thus, referring again to Figure 8, the conversation model 100 may include time attributes 106 which may include time ranges in a time state vector. Show attributes 104 may include a show set and selected show. The time attributes and show attributes are components of an utterance. Other components of the utterance may be "who said what" as indicated at 107 and immediate commands as indicated at 105. The conversation model may also include rules and methods 114 discussed herein as well as a history vector 46, dialog control 52 and a grammar 10a.



The methods and rules 114 in Figure 8 may include a number of methods used by the unit 10. For example, a method SetSelected() may be used by the unit 10 to tell the voice user interface 12 what shows have been selected by the graphical user interface 14. The method Speak() may be used to give other parts of the system, such as the graphical user interface 14, the ability to speak. If the synthesizer 20 is already speaking, then a Speak() request is queued to the synthesizer 20 and the method returns immediately. See specification at page 19, line 20 through page 20, line 22.

The method SpeakIfQuiet() may be used by the unit 10 to generate speech only if the synthesizer 20 is not already speaking. If the synthesizer is not speaking, the text provided with the SpeakIfQuiet() method may be given to the synthesizer 20. If the synthesizer is speaking, then the text may be saved, and spoken when the synthesizer is done speaking the current text.

## **VI. ISSUES**

- A. Is Claim 1 Anticipated by Haddock?**
- B. Is Claim 13 Obvious Over Haddock by Itself?**
- C. Is Claim 14 Obvious Over Haddock Taken Alone?**
- D. Is Claim 15 Obvious Over Haddock Taken Alone?**
- E. Is Claim 16 Obvious Over Haddock Taken Alone?**
- F. Is Claim 17 Obvious Over Haddock Taken Alone?**
- G. Is Claim 24 Obvious Over Haddock Taken Alone?**

## VII. GROUPING OF THE CLAIMS

For convenience on appeal, claims may be grouped as follows with the group representative claim indicated by underlining.

- Group A: Claims 1, 2, 3, 6, 7, 8, 9, 10
- Group B: Claims 13, 4, 5, 11, 12, 18, 23, 26, 27, 28, 29, 30
- Group C: Claims 14, 19, 25
- Group D: Claims 15, 20
- Group E: Claims 16, 21
- Group F: Claims 17, 22
- Group G: Claim 24

## VIII. ARGUMENT

### A. Is Claim 1 Anticipated by Haddock?

Claim 1 calls for developing a state vector representing the meaning of a spoken query and forming an attribute, value pair for the state vector. As explained herein at page 5, lines 3- and 10, a state vector is a representation of the meaning of an utterance by the user. The state vector may be made up of attribute, value pairs. See *infra* at p. 16, lines 14, *et seq.* For example, in the query “When is X-Files on this afternoon?” the request is the “when” part of the query. The request is an attribute whose value is “when”. Similarly, the query has a title attribute whose value is the “X-Files”. Thus, each attribute, value pair includes a name and a value.

In contrast in the cited patent to Haddock, no state vector is formulated and no attribute value pairs are created. In other words, the spoken language is not broken into state vectors and attribute, value pairs. In situations where the natural language may be so simplified, the use of

state vectors with attribute, value pairs considerably simplifies speech analysis. In contrast, in open ended situations, which is apparently the situation described in the Haddock reference, no such simplification is apparently possible.

Haddock never suggests using a state vector and certainly there is no attempt to break each state vector into attribute value pairs. In contrast, the entire question is broken into its semantic meaning, namely whether or not it includes nouns, phrases, verbs or sentences. Then, an attempt is made to determine what any pronouns mean by finding the pronouns and then looking to history to resolve their meaning. Again, there is never any attempt to set up state vectors or attribute, value pairs.

Therefore, the rejection of claim 1 should be reversed

**B. Is Claim 13 Obvious Over Haddock by Itself?**

Claim 13 calls for developing a first representation of a current user query. A second representation of a previous user query is developed and a determination is made whether the first representation includes only one of two types of variables and if so, the first representation is merged with the second representation. This claim is not specifically addressed in paragraph 1 of the final rejection. Certainly there is no developing of first and second representations of current and previous user queries and determining whether the first representation includes “only one of two types of variables” and if so, merging the first representation with the second representation.

In this regard, it is specifically stated in paragraph 4 of the first office action (paper no. 6, at paragraph 3) that “Haddock et al. do not specifically teach determining whether the utterance representation includes both types of variables”. Certainly, if this is so, the obviousness rejection of claim 13 based on Haddock alone is inappropriate.

Therefore, the rejection of claim 13 should be reversed.

**C. Is Claim 14 Obvious Over Haddock Taken Alone?**

Claim 14 is dependent on claim 13 and calls for storing instructions that cause a processor-based system to determine whether the first representation of a current user query includes only a where variable and, in such case, use a second representation of a previous user in query to form a third representation and insert the where variable into the second representation. Thus, claim 14 requires determining whether the first representation only includes a where variable and, in such case, using the second representation to form a third representation while inserting the where variable in the second representation.

Where variables represent information that the user is supplied. The where variable may define what the user has said. The where variable provides restrictions on the scope of what the user has asked for. See *infra* at p. 5, lines 3-12.

Referring to Figure 6, a check at diamond 72 determines whether both select and where variables are present. If so, no history is utilized as indicated in block 74. If not, a check at diamond 80 determines whether where variables only are included. If so, the where variable is merged with select in the history attributes as indicated in block 82. See also *infra* at p. 12, lines 12-14.

The final rejection never specifically addresses claim 14. Thus, it appears that there is no basis for the obviousness rejection. Certainly, the Haddock reference does not disclose anything even remotely similar to claim 14.

Therefore, the rejection of claim 14 should be reversed.

**D. Is Claim 15 Obvious Over Haddock Taken Alone?**

Claim 15, dependent on claim 13, relates to an article that determines whether the first representation has only a select variable, uses the second representation to form a third representation and inserts the select variable into the second representation.

Again, it does not appear that this claim was specifically addressed in the office action and, therefore, the rejection, based on a single reference, should be reversed.

Again, referring to Figure 6, at diamond 76, if it is determined that only a select variable is present, the where variable is merged with the history attributes as indicated in block 78.

Since this feature is nowhere disclosed in the cited reference, the rejection should be reversed.

**E. Is Claim 16 Obvious Over Haddock Taken Alone?**

Claim 16 calls for an article storing instructions that cause a processor-based system to determine whether neither a where or a select variable is contained in the first representation. The first representation is a representation of a current user query. If so, the third representation vector is made the same as the second representation. The second representation is a representation of a previous user query.

This claim can be better understood by referring to Figure 6. If neither a select or where variable is determined in diamond 76 and 80, then in block 84 the history is not changed. Thus, the third representation vector is made the same as the second representation. Again, no effort was made to address this claim in the office action and nothing in the Haddock reference suggests any such software.

Therefore, claim 16 should be allowed and the rejection should be reversed.

**F. Is Claim 17 Obvious Over Haddock Taken Alone?**

Claim 17, dependent on claim 13, calls for an article that stores instructions that cause a system to determine whether both a where and a select variable are contained in the first representation. If so, the first representation is used to form the third representation and the third representation is used as the second representation.

Referring to Figure 6, claim 17 would correspond to the situation where the answer at diamond 72 is yes, indicating that both select and where variables are present. In such case, no history is used as indicated in block 74. History is not needed to derive the in context meaning as indicated in block 74. See *infra* page 12, lines 6-8.

In the office action, the Examiner indicates that if the utterance contains both attributes of the utterance there is no ambiguous query and there is no need to use the history vector. But, of course, the cited Haddock reference does not even identify select or where variables. This is simply the totally unsupported conclusion of the Examiner having the benefit of hindsight. With the benefit of hindsight, and nowhere referring to the reference, the Examiner concludes that it would be obvious, without any support in the reference, to do what is claimed. But what is claimed involves the determination of a specific set of circumstances. The reference does not even concern itself with select and where variables and whether or not both variables are present in the utterance. Therefore, the reference cannot possibly provide a *prima facie* rejection.

Here, a single reference is cited and obviousness is concluded, although it is apparently admitted that the reference does not meet the claim limitations. The Examiner is required to provide, from the prior art, a rationale to modify the cited reference to meet the claim limitations. Here, the Examiner simply concludes, with little or no support from the reference, that the invention is obvious.

Therefore, the rejection should be reversed.

**G. Is Claim 24 Obvious Over Haddock Taken Alone?**

Claim 24 is dependent on claim 23. Claim 24 adds the limitation that the software develops a state vector representing the meaning of a spoken query. The state vector is formed of attribute value pair with a non-recursive data structure as said value. As pointed out at page 16 *infra*, lines 3-6, if each attribute value pair includes a name and a value, the data structure may be simplified by ensuring that the values are simple structures such as integers, strings, lists, or other database records as opposed to another state vector. Thus, by using non-recursive data structures, a simpler system may be implemented. There is no discussion of such structures in the cited Haddock reference, which is cited as the sole Section 103 rejection.

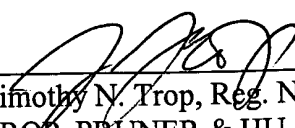
Therefore, the rejection of claim 24 should be reversed.

**IX. CONCLUSION**

Applicant respectfully requests that each of the final rejections be reversed and that the claims subject to this Appeal be allowed to issue.

Respectfully requested,

Date: 8/7/2002

  
\_\_\_\_\_  
Timothy N. Trop, Reg. No. 28,994  
TROP, PRUNER & HU, P.C.  
8554 Katy Freeway, Ste. 100  
Houston, TX 77024  
713/468-8880 [Phone]  
713/468-8883 [Fax]

## APPENDIX OF CLAIMS

The claims on appeal are:

1. An article comprising a medium for storing instructions that cause a processor-based system to:  
  
develop a state vector representing the meaning of a spoken query; and  
  
form an attribute, value pair for said state vector.
2. The article of claim 1 further storing instructions that cause a processor-based system to develop an utterance vector from a current user query and a history vector from a previous user query.
3. The article of claim 2 further storing instructions that cause a processor-based system to merge the utterance vector with the history vector to develop an in-context meaning vector.
4. The article of claim 3 further storing instructions that cause a processor-based system to determine whether the utterance vector includes only one type of variable, a first or a second of two variable types, and if so, merge the variable with the history vector to derive said in-context meaning vector.
5. The article of claim 4 further storing instructions that cause a processor-based system to determine whether the utterance vector includes both the first and second variable types and if so to refrain from using the history vector to derive said in-context meaning vector.



6. A method comprising:  
developing a state vector that represents the meaning of a spoken query; and  
form an attribute, value pair for said state vector.
7. The method of claim 6 wherein using a non-recursive data structure includes using only non-recursive data structures as said value.
8. The method of claim 6 including refraining from using another state vector as said value.
9. The method of claim 6 including developing an utterance vector from a current user query and a history vector from a previous user query.
10. The method of claim 9 including merging the utterance vector with the history vector to develop in-context meaning vector.
11. The method of claim 10 including determining whether the utterance vector includes only one of two types of variables, and if so, merging the variable with the history vector to derive said in-context meaning vector.
12. The method of claim 11 including determining whether the utterance vector includes both the first and second variable types and if so refrain from using said history vector to derive said in-context meaning vector.

13. An article comprising a medium for storing instructions that cause a processor-based system to:

develop a first representation of a current user query;

develop a second representation of a previous user query; and

determine whether the first representation includes only one of two types of variables, and if so, merge the first representation with the second representation to form a third representation.

14. The article of claim 13 further storing instructions that cause a processor-based system to determine whether the first representation includes only a where variable and in such case use the second representation to form a third representation and insert the where variable into the second representation.

15. The article of claim 13 further storing instructions that cause a processor-based system to determine whether the first representation has only a select variable, use the second representation to form a third representation and insert the select variable into the second representation.

16. The article of claim 13 further storing instructions that cause a processor-based system to determine whether neither a where or a select variable is contained in the first representation and in such case to make the in-third representation vector the same as second representation.

17. The article of claim 13 further storing instructions that cause a processor-based system to determine whether both a where variable and a select variable are contained in the first representation and if so, use the first representation to form the third representation and use the third representation as the second representation.

18. A method comprising:  
developing a first representation from a current user query;  
developing a second representation from a previous user query; and  
determining whether said first representation includes only one of two variable types and if so, merging the first representation with the second representation to form the third representation.

19. The method of claim 18 including determining whether the first representation includes only a where variable and in such case using the second representation as the third representation and inserting the where variable into the second representation.

20. The method of claim 18 including determining whether the first representation has only a select variable and if so, using the second representation as the third representation and inserting the select variable into the second representation.

21. The method of claim 18 including determining whether neither a where or a select variable is contained in the first representation and in such case, making the third representation the same as the second representation.

22. The method of claim 18 including determining whether both a where variable and a select variable are contained in the first representation and if so using the first representation to form the third representation and using the third representation as the second representation.

23. A system comprising:  
a processor; and  
a storage coupled to said processor, said storage storing software that develops a first representation from a current user query, develops a second representation from a previous user query, determines whether the first representation includes only one of two variable types and if so merges the first representation with the second representation to form a third representation.

24. The system of claim 23 wherein said software develops a state vector representing the meaning of a spoken query, said state vector formed of a attribute, value pair with a non-recursive data structure as said value.

25. The system of claim 23 wherein said software determines whether the first representation includes only a where variable and in such case, uses the second representation as the third representation and inserts the where variable into the second representation.

26. The system of claim 23 including a speech recognizer and a speech synthesizer communicating with said software.

27. The system of claim 26 including a graphical user interface stored in said storage and synchronized to said software.

28. The system of claim 23 including an electronic programming guide application, said software creating a conversational speech responsive system.

29. The system of claim 23 wherein said system is a set-top box controlling a television receiver and implementing an electronic programming guide.

30. The system of claim 29 including a remote control unit coupled to said set-top box through a wireless interface.

**TRANSMITTAL OF APPEAL BRIEF (Large Entity)**Docket No.  
ITL.0343USIn Re Application Of: **Christopher H. Genly**Serial No.  
09/494,796Filing Date  
January 31, 2000Examiner  
Angela ArmstrongGroup Art Unit  
2654Invention: **PROVIDING INFORMATION IN RESPONSE TO SPOKEN REQUESTS****RECEIVED****AUG 14 2002**

Technology Center 2600

**TO THE ASSISTANT COMMISSIONER FOR PATENTS:**

Transmitted herewith in triplicate is the Appeal Brief in this application, with respect to the Notice of Appeal filed on June 26, 2002.

The fee for filing this Appeal Brief is: **\$320.00**

- ☒ A check in the amount of the fee is enclosed.
- ☐ The Commissioner has already been authorized to charge fees in this application to a Deposit Account. A duplicate copy of this sheet is enclosed.
- ☒ The Commissioner is hereby authorized to charge any fees which may be required, or credit any overpayment to Deposit Account No. **20-1504**  
A duplicate copy of this sheet is enclosed.

**RECEIVED**  
2002 AUG 12 PM 2:24  
BOARD OF PATENT APPEALS  
AND INTERFERENCESDated: 8/7/02  
Signature

**Timothy N. Trop, Reg. No. 28,994**  
**Trop, Pruner & Hu, P.C.**  
**8554 Katy Freeway, Suite 100**  
**Houston, Texas 77024**  
**(713) 468-8880**  
**(713) 468-8883 (fax)**

I certify that this document and fee is being deposited on August 7, 2002 with the U.S. Postal Service as first class mail under 37 C.F.R. 1.8 and is addressed to the Assistant Commissioner for Patents, Washington, D.C. 20231.

  
Signature of Person Mailing Correspondence**Cynthia L. Hayden**

Typed or Printed Name of Person Mailing Correspondence

CC:

**TRANSMITTAL OF APPEAL BRIEF (Large Entity)**Docket No.  
ITL.0343US

In Re Application Of: Christopher H. Genly

Serial No.  
09/494,796Filing Date  
January 31, 2000Examiner  
Angela ArmstrongGroup Art Unit  
2654


Invention: PROVIDING INFORMATION IN RESPONSE TO SPOKEN REQUESTS

TO THE ASSISTANT COMMISSIONER FOR PATENTS:

Transmitted herewith in triplicate is the Appeal Brief in this application, with respect to the Notice of Appeal filed on June 26, 2002.

The fee for filing this Appeal Brief is: \$320.00

- ☒ A check in the amount of the fee is enclosed.
- ☐ The Commissioner has already been authorized to charge fees in this application to a Deposit Account. A duplicate copy of this sheet is enclosed.
- ☒ The Commissioner is hereby authorized to charge any fees which may be required, or credit any overpayment to Deposit Account No. 20-1504  
A duplicate copy of this sheet is enclosed.

  
Signature

Timothy N. Trop, Reg. No. 28,994  
Trop, Pruner & Hu, P.C.  
8554 Katy Freeway, Suite 100  
Houston, Texas 77024  
(713) 468-8880  
(713) 468-8883 (fax)

Dated: 8/7/02

RECEIVED  
2002 AUG 12 PM 2:24  
BOARD OF PATENT APPEALS  
AND INTERFERENCES

I certify that this document and fee is being deposited on August 7, 2002 with the U.S. Postal Service as first class mail under 37 C.F.R. 1.8 and is addressed to the Assistant Commissioner for Patents, Washington, D.C. 20231.

  
Signature of Person Mailing Correspondence

Cynthia L. Hayden

Typed or Printed Name of Person Mailing Correspondence

CC: